GRPC-DEMO

מדריך זה יסביר בקצרה כיצד ניתן לבנות אפליקציות שמדברות בניהן בעזרת GRPC

האפליקציות בדמו זה נכתבו בשפת Java-script אך ניתן לכתוב את האפליקציות בכל שפה נתמכת

מדריך להפעלה

- Windows systems
  1. התקנת Nodejs & npm, מדריך להתקנה https://www.youtube.com/watch?v=X-FPCwZFU_8
- Linux Systems(Debian)
  1. התקנת Nodejs & npm, מדריך להתקנה https://www.youtube.com/watch?v=K6QiSKy2zoM

1. יצירת פרויקט בJS

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

┌──(kali㉿kali)-[~/Desktop/New Folder]
└─$ cd grpc-demo


┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
```

2. יצירת קובץ proto המציין את הפונקציות השירות, מתודות וכו'

```
grpc-demo > todo.proto > ...
// specifying syntax version
syntax = "proto3";


package todoPackage;


// Creating Service (The SCHEMA)
service Todo {
    // Creating TODO methods
    rpc createTodo(TodoItem) returns (TodoItem);
    // voidNoParam == VOID, (there's no void in protobuf)
    rpc readTodos(voidNoParam) returns (TodoItems);
    rpc readTodosStream(voidNoParam) returns (stream TodoItem);

}

// Dummy message
message voidNoParam {}

// (The SCHEMA)
message TodoItem {
    int32 id = 1;
    string text = 2;
}
// (The SCHEMA)
message TodoItems {
    // Array
    repeated TodoItem items = 1;
}
```

3. ניתן לקמפל קובץ ה-proto לקובץ JS בשני דרכים:
   - בעזרת פקודה: (פקודה זו רלוונטית רק לjs, לכל שפה נתמכת ב-protocolbuf הפקודה משתנה)
   - בעזרת חבילות עזר לכל שפה, לJS משתמשים ב"@grpc/proto-loader"
4. יצירת שרת באפליקציה והתקנות של חבילות נחוצות

```
todo.proto    JS server.js ×    JS client.js
grpc-demo > JS server.js > ...

// js grpc package
const grpc = require("grpc");
// js protocolbuf compiler
const protoLoader = require("@grpc/proto-loader")

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$ npm i grpc @grpc/proto-loader
```

5. טעינת החבילה שמצוינת בקובץ ה-Proto כדי להשתמש בשירות.

```
// Loading sync the package from proto file
const packageDef = protoLoader.loadSync("todo.proto", {});
// Creating grpc object from package
const grpcObject = grpc.loadPackageDefinition(packageDef);

const todoPackage = grpcObject.todoPackage;
```

6. יצירת שרת תוך ציון כתובת ופורט בנוסף מכיוון ש-protocolbuf עובד עם HTTP2 צריך פרטי התחברות אך לצורך ההדגמה השתמשנו במצב "ללא הפרטים מזהים"

```
// Creating grpc Server
const server = new grpc.Server();
// Binding grpc Server to address and port.
// Creating the server without credentials
server.bind("0.0.0.0:40000",
 grpc.ServerCredentials.createInsecure());
```

7. הוספת השירות לשרת, מיפוי הפונקציות של השירות(עד כאן השרת לא ידע כלל על השירות והמתודות שיש בתוכו) והפעלת השרת

```
// Adding the service from the package to the server
server.addService(todoPackage.Todo.service,
    {
        "createTodo": createTodo,
        "readTodos" : readTodos,
        "readTodosStream": readTodosStream
    });
server.start();
```

8. יצירת משימות בשרת ושמירתם במערך זמני(לצורך ההדגמה, כאן ניתן להשתמש בDB)

```
// Creating TODO'S
function createTodo (call, callback) {
    const todoItem = {
        // init simple id's
        "id": todos.length + 1,
        // Getting the TODO'S from the client
        "text": call.request.text
    }
    todos.push(todoItem)
    callback(null, todoItem);
}
```

9. החזרת המשימות כאובייקט stream (הדרך היעילה)

```
// Sending to the client the TODO'S as stream
function readTodosStream(call, callback) {

    todos.forEach(t => call.write(t));
    call.end();
}
```

10. החזרת המשימות כמערך (הדרך הפחות היעילה)

```
// Sending to the client the TODO'S as array (less efficient)
function readTodos(call, callback) {
    callback(null, {"items": todos})
}
```

11. יצירת צרכן לשירות, לצורך הגדרת הלקוח הלקוח אמור לדעת על השירות מפרוקוטול באפר ולכן ההגדרה כמו בשרת

todo.proto | server.js | client.js

rpc-demo > client.js > ...

```
const grpc = require("grpc");
const protoLoader = require("@grpc/proto-loader")
const packageDef = protoLoader.loadSync("todo.proto", {});
const grpcObject = grpc.loadPackageDefinition(packageDef);
const todoPackage = grpcObject.todoPackage;

```

12. הגדרת קבלת מידע מהמשתמש כארגומנט

```
const text = process.argv[2];
```

13. חיבור הלקוח לשרת

```
// Connecting to server
const client = new todoPackage.Todo("localhost:40000",
grpc.credentials.createInsecure())
console.log(text)
```

14. יצירת משימה (המזהה הינו מידע שגוי, המזהה נקבע ע"י השרת)

```
// Creating TODO
client.createTodo({
    "id": -1,
    "text": text
}, (err, response) => {

    console.log("Recieved from server " + JSON.stringify(response))

})
```

15. לצורך קבלת המידע מהשרת הלקוח יכול לקבל בשני דרכים

- ע"י קבלת המידע כבלוק של כלל המידע (יכול להמשך המון זמן במערכים גדולים)

```
// the WRONG way, take's a lot to process big date

/*
client.readTodos(null, (err, response) => {
    console.log("read the todos from server " + JSON.stringify(response))
    if (!response.items)
        response.items.forEach(a=>console.log(a.text));
})
*/

```

- ע"י אובייקט סטרים המחלק את המידע ושולח בחלקים

```
// Read TODO'S as stream
const call = client.readTodosStream();
call.on("data", item => {
    console.log("received item from server " + JSON.stringify(item))
})
```

דוגמת הרצה

1. הרצת השרת בעזרת nodeJS

```
todo.proto    server.js    client.js
grpc-demo > server.js > packageDef

// Loading sync the package from proto file
const packageDef = protoLoader.loadSync("todo.proto", {});
// Creating grpc object from package
const grpcObject = grpc.loadPackageDefinition(packageDef);

const todoPackage = grpcObject.todoPackage;

// Creating grpc Server
const server = new grpc.Server();
// Binding grpc Server to address and port.
// Creating the server without credentials
server.bind("0.0.0.0:40000",
 grpc.ServerCredentials.createInsecure());

// Adding the service from the package to the server
server.addService(todoPackage.Todo.service,
    {
        "createTodo": createTodo,
        "readTodos" : readTodos,
        "readTodosStream": readTodosStream
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$ node server.js
```

2. הרצת הלקוח בnode אחר בעזרת nodeJS, בנוסף הלקוח שולח את המשימה שאותה הוא רוצה לשמור.

```
todo.proto    server.js    client.js
grpc-demo > server.js > packageDef
6
7   // Loading sync the package from proto file
8   const packageDef = protoLoader.loadSync("todo.proto", {});
9   // Creating grpc object from package
10  const grpcObject = grpc.loadPackageDefinition(packageDef);
11
12  const todoPackage = grpcObject.todoPackage;
13
14  // Creating grpc Server
15  const server = new grpc.Server();
16  // Binding grpc Server to address and port.
17  // Creating the server without credentials
18  server.bind("0.0.0.0:40000",
19   grpc.ServerCredentials.createInsecure());
20
21  // Adding the service from the package to the server
22  server.addService(todoPackage.Todo.service,
23      {
24          "createTodo": createTodo,
25          "readTodos" : readTodos,
26          "readTodosStream": readTodosStream

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$ node client.js "go shopping"
```

3. הרצת לבסוף מתקבלת התשובה מהשרת של כלל המשימות השמורות בעזרת אובייקט stream
   - דוגמה 1 עם המשימה "ללכת לקניות"

```
┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$ node client.js "go shopping"
go shopping
server done!
Recieved from server {"id":1,"text":"go shopping"}

┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$
```

   - דוגמה 2 עם המשימה לעשות ש"ב

```
┌──(kali㉿kali)-[~/Desktop/New Folder/grpc-demo]
└─$ node client.js "do homework"
do homework
received item from server {"id":1,"text":"go shopping"}
Recieved from server {"id":2,"text":"do homework"}
server done!
```